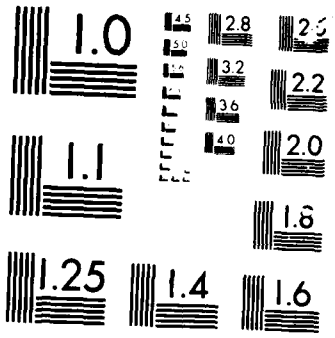
AD-A169 293 GAUSSIAN ELIMINATION ON HYPERCUBES(U) YALE UNIV NEW HAVEN CT DEPT OF COMPUTER SCIENCE Y SAAD MAR 86 YALEU/DCS/RR-462 N00014-82-K-0184 1/1

UNCLASSIFIED F/G 9/2 NL



MICROCOPY

(13)

DTIC
ELECTE
JUL 01 1986
D

AD-A169 293

Gaussian elimination on hypercubes

Youcef Saad

Research Report YALEU/DCS/RR-462
March 1986

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

DTIC FILE COPY

YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

86 5 9 042

(13)

Abstract. We compare several implementations of the Gaussian elimination algorithm for solving dense linear systems on hypercube parallel processors. We distinguish between two classes of methods: methods that require to move the elimination row (or column) to all processors before the elimination proceeds, and methods that require only moving data to nearest neighbors. Algorithms of the second class, which we call pipelined algorithms, require only a ring or grid structure which is embedded into the hypercube. One of our main conclusions is that for Gaussian elimination the additional connectivity of the hypercube topology over that a two-dimensional grid of processors does not help much in improving efficiency. Another result of our analysis is that there is little reason for using row or column type algorithms instead of grid algorithms. One of the goals of the paper is also to show a simple model of complexity analysis at work, by comparing the estimated times that it provides with the actual execution times.

DTIC
ELECTE
JUL 0 1 1986
S D
D

Gaussian elimination on hypercubes

Youcef Saad

Research Report YALEU/DCS/RR-462
March 1986

Prepared for the international workshop on parallel algorithms and architectures. Luminy, France, April 14-18 1986.

This work was supported in part by ONR grant N00014-82-K-0184, and in part by a joint study with IBM Kingston

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

1. Introduction

This paper considers the implementation of a standard method, namely Gaussian elimination, on a multiprocessor based on the hypercube topology. Our objective is not to present new and better algorithms, but rather to analyze the behavior of two different implementations of Gaussian elimination in a hypercube based architecture. Suppose that the rows (or columns) of the system are distributed in some way among the processors. At each step j of the first implementation, which we call a broadcast algorithm, the elimination row (or column) is sent to all processors that hold at least one of the rows $i+1, i+2, \ldots, N$. This data movement is a broadcast type operation : one processor sends the same data to all others or to a subset of all others. Once the elimination row (or column of multipliers) has been broadcast, the arithemtic corresponding to the i^{th} step of Gaussian elimination is performed.

The second approach differs only in its organization. Processors are no longer required to perform the j^{th} step of Gaussian elimination at the same time (or approximately at the same time). A processor awaits for the j^{th} row, passes it to the next processor as soon as it arrives and then proceeds with the arithmetic. The next processor will in turn pass the row to a neighbor and proceed with arithmetic. This idea of pipelining Gaussian elimination is an old idea which is predominant in particular in systolic architectures, [7, 14, 15].

These two approaches are well-known and there is an extensive literature on their performance and their implementations on various architectures, see for example [9, 8, 5, 6, 4, 15, 13, 19, 20]. Both of them can be implemented on hypercubes, but the second only requires a one-dimensional or a two-dimensional grid of processors. The question then arises: can we expect the broadcast methods, which take advantage of the complex hypercube topology, to outperform the pipelined methods? As will be seen if the matrix is mapped by stripes, i.e., a few rows or columns per processor, then the answer is clear-cut: there is no compelling reason for using a broadcast algorithm.

Similar broadcast and pipelined algorithms can also be derived for grid mappings, in which the matrix is split into squares blocks that are mapped to the nodes of a processor grid embedded in a hypercube. In case these grid mappings are used, and no pivoting is needed, then again pipelined methods are superior. When pivoting is needed a solution of choice appears to be a pairwise pivoting technique, see [4, 20, 22].

This paper will present a few techniques based on both ring and grid mappings and will provide the estimated exection times for each algorithm. Numerical experiments will be proposed to verify the accuracy of the model and then a conclusion based on the estimated execution times will be drawn.

2. Hypercube topology: properties and assumptions

Hypercube multiprocessors have been very successful in the recent years among academic institutions as well as industrial research groups. There are currently several comercially available parallel processors based on the binary n-cube network. For example, we can mention Intel's iPSC with up to 128 processors, Ametek's System 14 with 256 processors, Ncube's NCUBE-10 with 1024 processors, and Thinking Machines' Connection Machine with 64000 processors. It is likely that many others will soon appear in the market.

By definition, an n-dimensional hypercube, or binary $n-$cube, consists of $k \equiv 2^n$ nodes numbered by n-bit binary numbers, from 0 to $2^n - 1$ and interconnected so that there is a link between two processors if and only if their binary labels differ by exactly one bit. An alternative definition which is perhaps more helpful in undestanding the nature of the topology, is to define the hypercube in a recursive fashion:

- A zero-cube consists of only one node;

Availability Codes
Dist | Avail and/or Special
A-1

- To obtain an $(n+1)-$ cube take two identical n-cubes and link their corresponding nodes in a one-to-one fashion.

This is illustrated in Figure 1. Thus, for the case $n = 3$, the 8 nodes are simply at the vertices of a three-dimensional cube. In an n-cube each node has n neighbors with which it can communicate. For further details see the references [2, 16, 17, 21] and the references therein.

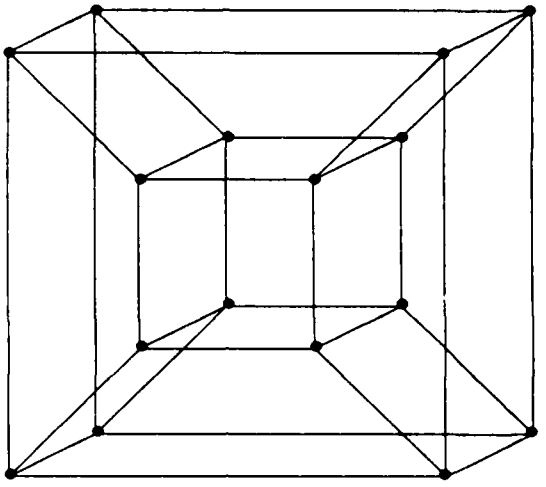


Figure 1: A Hypercube of Dimension 4.

One of the main advantages of hypercubes, perhaps the most important one, is that many of the classical topologies such as two-dimensional or three-dimensional meshes, rings, trees, can be embedded preserving proximity in them [11, 3, 16, 21, 10, 1]. An important consequence of this is that we can design algorithms for ring or grid structures and be able to map them on hypercubes. This allows us to choose the best alternative among all of the possible choices. In the present paper, we will show how to exploit this feature when implementing a simple algorithm such as Gaussian elimination.

Some properties of hypercubes have been developed in [16] and communication problems in the hypercube have been examined in detail in [17]. We would like to summarize some of the properties that will be useful in the remainder of this paper. For details see [16].

An obvious property is that the n-cube is a connected graph of diameter n. It is usual to number the nodes of an n-cube by binary numbers from 0 to $2^n - 1 = k - 1$ according to the definition of an n-cube, i.e., so that two labels of any two neighbors differ only in one bit. We will denote by $H(X, Y)$ the Hamming distance between two binary numbers X and Y, i.e., the number of bits that differ between X and Y.

A particularly important notion for the hypercube topology is that of Gray codes. A Gray code of dimension n is a finite sequence $\{g_0^{(n)}, g_1^{(n)}, \ldots g_{2^n-1}^{(n)}\}$ of 2^n binary numbers which represent all n-bit binary numbers and so that

$$H(g_i^{(n)}, g_{i+1}^{(n)}) = 1, \quad \forall\, i$$

where the subscripts should be considered modulo 2^n. Thus, a Gray code defines a sequence of all the nodes of a hypercube so that any two successive nodes in the sequence are neighbors: in graph

theory terminology this is a Hamiltonian circuit. Therefore we can embed a ring of 2^n nodes into a hypercube. Throughout the paper when we refer to a ring we will always mean a ring embedded in the hypercube in this manner.

Gray codes can also be used to map two-dimensional or three-dimensional grids into hypercubes. For example if we want to embed an 2^{n_1} by 2^{n_2} two-dimensional grid into an n-cube with $n_1 + n_2 = n$ it suffices to map the node (i, j) of the grid, where $0 \leq i \leq 2^{n_1} - 1$, and $0 \leq j \leq 2^{n_2} - 1$, into the node of the hypercube whose binary label is

$$g_i^{(n_1)} \wedge g_j^{(n_2)}$$

where $\wedge$ denotes concatenation. Observe that the set of nodes obtained by fixing one coordinate and letting the other vary, forms a subcube. In other words every row or column of processors of the embedded processor grid forms a subcube of the cube. Again, in this paper a grid will always mean a grid embedded in an n-cube in the manner just described.

We should briefly mention how Gray codes can be generated. Let $G_n = \{g_0, g_2, \ldots, g_{2^n-1}\}$ be the n-bit Gray code, with $G_1 \equiv \{0, 1\}$ and denote by G_n^R the sequence obtained from G_n by reversing its order, and by $0G_n$ (resp. $1G_n$) the sequence obtained from G_n by prefixing a zero (resp. a one) to each element of the sequence. Then Gray codes of arbitrary order can be generated by the recursion:

$$G_{n+1} = \{0G_n, 1G_n^R\}, \quad n = 1, \ldots \tag{2.1}$$

For example,

$$G_2 = \{00, 01, 11, 10\},$$

$$G_3 = \{000, 001, 011, 010, 110, 111, 101, 100\}. \tag{2.2}$$

The particular Gray code generated by the above algorithm is called the ***binary reflected Gray code***.

Except for the Connection Machine which is SIMD and bit - serial, the currently existing hypercubes function in MIMD message passing mode: a processor executes its own program which runs independently from the programs of other nodes, except that it will sometimes require data that is provided by processes running on other nodes. Thus, synchronization is achieved only through availability of data: when an operand is needed the processor waits until it is available before pursuing the execution of its program.

In order to estimate the total time that is required to execute a given algorithm, we will use a simple model for timing both arithmetic and interprocessor communication. For arithmetic we will assume that it takes the time ω to execute a pair of operations consisting of an addition and a multiplication. Performing m such pairs will require the time $m\omega$.

For communication, we assume that it takes the time

$$\beta + m\tau, \tag{2.3}$$

to transfer m words from one processor to any of its n neighbors, where β is the communication start-up in seconds and τ is the elemental transfer time, in seconds per word. Moreover, communication in all the n channels can take place simultaneously: a node can do either a receive from or a send to any neighbor while doing a receive or a send with another neighbor. This assumption is important because it implies that the total communication speed from a node is proportional to its fanout and therefore that the hardware can be effectively utilized.

In some of the Gaussian elimination algorithms we need to move the same data, i.e., a row or a column, from a given node to all other nodes. This is called a broadcast operation. A standard algorithm to broadcast a data packet D from node 0^n to all other nodes in an n-cube is the following.

ALGORITHM: Hypercube Broadcast

In every node L do:

For $j = 1, 2, 3, \ldots n$ do:

If $L < 2^{j-1}$ move D to node $L + 2^{j-1}$.

This basic algorithm is not optimal. If D consists of N words it will take an approximate time of

$$n\,[N\tau + \beta] = (Log_2 k)\,[N\tau + \beta] \tag{2.4}$$

to execute the above algorithm. To improve efficiency, one can partition the data set D into ν equal packets and pipeline the data packets in succession. There is an optimal number of packets and the corresponding optimal time to broadcast N words in an n-cube is [17] :

$$t_{opt}(N, n) \approx \left(\sqrt{N\tau} + \sqrt{(n-1)\beta}\right)^2. \tag{2.5}$$

However, in this paper we will use the simple upper bound given by the following inequality

$$t_{opt}(N, n) \leq 2(N\tau + n\beta) = 2(N\tau + \beta Log_2 k). \tag{2.6}$$

This upper bound is within a factor of about 2 of the actual best time (2.5).

3. Row and column oriented algorithms

3.1. Broadcast Row Algorithm

The simplest way to implement Gaussian elimination in parallel is to subdivide the matrix A into k blocks of $\frac{N}{k}$ rows each and assign one block to each processor of the ring successivily. If $P_0, P_1, \ldots P_{k-1}$ are the nodes of the hypercube, in a natural order, then let processor P_i hold rows $i\frac{N}{k} + 1$ to $(i+1)\frac{N}{k}$ of A and the corresponding components of the right hand side vector b, as indicated in Figure 2.

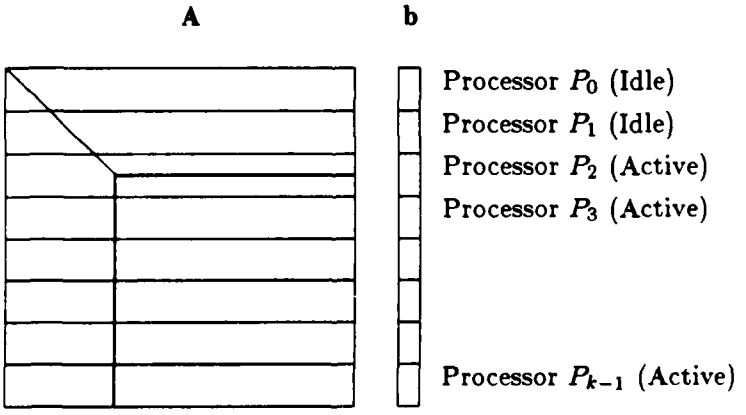


Figure 2: Gaussian Elimination on a Block Row Partitioned Matrix.

At step j, row j must be sent from the processor where it is located, say processor P_i, to processors $P_{i+1} \ldots P_{k-1}$ in order to perform the eliminations in each of them. From Section 2 we know that transferring a row of length $N - j + 1$ to all processors requires a time bounded from above by

$$2\,(\beta Log_2 k + (N - j + 1)\tau). \tag{3.1}$$

Here we do not have to move the row to all processors but only the processors $P_{i+1}, \ldots, P_{k-1}$, i.e., to $k - i - 1$ processors. Since the data is initially arranged so that block i is in Processor P_{i-1}, then it is easy to see that the processors $P_i, P_{i+1}, P_{i+2}, \ldots P_{k-1}$ are in a subcube of dimension $\lceil Log_2(k-i) \rceil$ and therefore we can perform a limited broadcast in $\lceil Log_2(k-i) \rceil$ steps. The communication time for each step is then bounded from above by

$$2\,(\lceil Log_2(k-i) \rceil \beta + (N-j)\tau) \leq 2\,((Log_2(k-i)+1)\beta + (N-j)\tau).$$

Summing up for $j = 1, \ldots N$:

$$2\frac{N}{k}\sum_{i=0}^{k-1} \lceil Log_2(k-i) \rceil \beta + 2\sum_{j=1}^{N-1}(N-j)\tau \leq 2\frac{N}{k}(k + Log_2(k!))\beta + 2\sum_{j=1}^{N-1}(N-j)\tau.$$

From Stirling's formula we get the approximate communication time

$$t_C \approx N(1 + Log_2\frac{k}{e})\beta + N^2\tau = N(Log_2\frac{2k}{e})\beta + N^2\tau.$$

To determine the time for arithmetic, we note that at step j a processor performs at most $\frac{N}{k}$ eliminations which require the time

$$\frac{N}{k}(N - j + 1)\omega. \tag{3.2}$$

Summing up over $N - 1$ steps, we obtain the estimated arithmetic time

$$t_A \approx \frac{N^3}{2k}\omega \tag{3.3}$$

Hence the total estimated time for this algorithm is

$$T_R \approx \frac{N^3}{2k}\omega + 2N Log_2\frac{k}{e}\beta + N^2\tau. \tag{3.4}$$

Notice the deterioration in efficiency in the arithmetic time due to the denominator $2k$ instead of the ideal $3k$ in the ω term of (3.4). The reason for this loss of efficiency is that at the end of the elimination process many processors, in fact most of them at the very end, are idle. A remedy is to interleave the rows with respect to processors, an idea used in [9]. The corresponding mapping is illustrated in Figure 3.

To estimate the communication time for this new version of the algorithm, we only observe that the difference with the previous algorithm is that a row must now be sent to all processors at every step (except the very last k steps but the corresponding difference is negligible.) Thus communication contributes to each step of the algorithm a time bounded from above by the expression

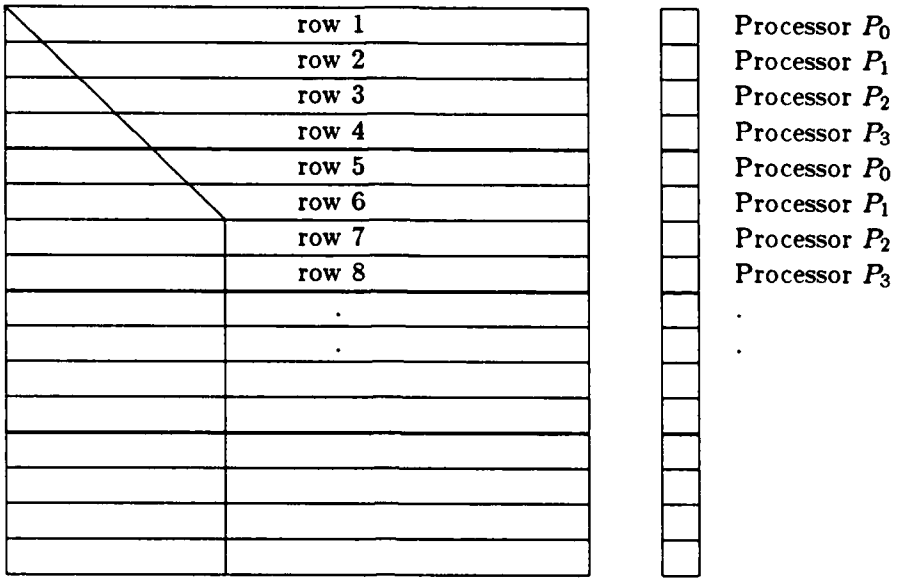


Figure 3: Gaussian Elimination with Interleaved Rows for $k = 4$.

(3.1), whose total over the $N-1$ steps of Gaussian elimination is about $N^2\tau + 2N Log_2(k)\beta$. Concerning arithmetic, observe that at step j, a processor typically performs the linear combination of row j with $\lceil (N-j)/k \rceil$ rows. Therefore, each step will now cost

$$\lceil \frac{N-j}{k} \rceil (N-j+1)\omega, \tag{3.5}$$

instead of (3.2). Defining the function

$$f(N,k) \equiv \sum_{i=1}^{N-1} \lceil \frac{i}{k} \rceil (i+1), \tag{3.6}$$

we get the estimated time

$$T_{R,B} \equiv f(N,k)\omega + N^2\tau + 2N\ (Log_2 k)\beta. \tag{3.7}$$

Note that if the integer division of $N-1$ by k is $N-1 = ks + r$ then we have

$$f(N,k) = sk\left[\frac{(s+1)(k+3)}{4} + \frac{k(s^2-1)}{3}\right] + \frac{r(s+1)}{2}(2N-r+1).$$

When $k << N$, then $f(N,k)$ can be approximated by $N^3/(3k)$ which yields

$$T_{R,B} \approx \frac{N^3}{3k}\omega + N^2\tau + 2N\ (Log_2 k)\beta \quad \text{for} \quad k << N. \tag{3.8}$$

Notice that a loss of efficiency takes place when $k \approx N$. In fact, for $k = N$ we obtain that $f(N,k) = \frac{1}{2}(N-1)(N+2)$, which means that the algorithm has an *arithmetic* efficiency of

approximately 2/3. By arithmetic efficiency we mean the efficiency that would be obtained in an ideal situation where communication times were negligible.

We should also point out that the above scheme can be generalized into a column oriented scheme in a straighforward manner. One of the reasons why column schemes have often been preferred to row oriented schemes is that row interchange is easy to implement with them. However, they have the disadvantage of leading to more inefficient triangular systems solutions [9]. Moreover, note that for row schemes column interchange can be used instead of row interchange.

3.2. Pipelined Ring Algorithm

In the pipelined ring algorithm, the computation is rearranged so that only nearest neighbor communication is required. We consider the general case where the number of processors k is larger than N, in fact ideally much larger, and assume the interleaved distribution illustrated in Figure 3. Here we denote by $P_0, P_1, \ldots P_{k-1}$ the sequence of processors that form a ring embedded in the hypercube using the Gray code embedding of section 2. The idea of the algorithm is that every processor executes the $N-1$ successive elimination steps of Gaussian elimination on the rows that it holds. To do so it must receive and store the pivot row, send it immediatly to the next processor in the ring, and then proceed with the elimination. Schematically the algorithm is as follows.

For $j = 1, ..., N-1$, do in every processor P_i:

1. Receive the j^{th} row from processor P_{i-1} and store it.
2. Send the j^{th} row just received to the processor P_{i+1}
3. Perform the elimination step number j for all rows $m, m>j$ that belong to processor P_i.

It might seem more natural at first to interchange 2 and 3, i.e., to compute as soon as the row is available and send the row to the next processor after the arithmetic is completed but this is not as efficient since the next processor will be idle waiting for the row to arrive while the Processor P_i is computing.

Let us now estimate the execution time of this a'gorithm. To this end we look at the block consisting of the k last rows of the system, see Figure 3. Without loss of generality we will make the additional assumption that the size N of the matrix is a multiple of k, the number of processors. The key observation is that the last row of the linear system, which is held in the last processor P_{k-1}, is the critical row in the sense that the job will be completed when the last step of Gaussian elimination will be performed on this last row.

Row 1 will reach the last processor P_{k-1} after crossing $k-1$ processor, i.e., after the time

$$(k-1)[\beta + N\tau] \tag{3.9}$$

Thereafter, processor P_{k-1} will use it for the elimination and then receive the next row which should be ready to be moved from processor P_{k-2}. Thus, each step j requires the linear combination of the row j with $\lceil (N-j)/k \rceil$ rows, namely all those rows among rows $j+1, j+2, \ldots, N$ that are held in processor P_{k-1}. This is followed by a communication task to receive the next row which is of length $N-j$. Therefore, in Processor P_{k-1} each step j of Gaussian elimination will require the time

$$\lceil \frac{N-j}{k} \rceil (N-j+1)\omega + (N-j)\tau + \beta.$$

Summing up these times over $j = 1, \ldots, N-1$ and adding the result to the delay time (3.9), we get by use of the definition (3.6) and by dropping lower order terms:

$$T_{P,R} = f(N,k)\omega + \frac{1}{2}N(N+2k)\tau + (N+k)\beta \tag{3.10}$$

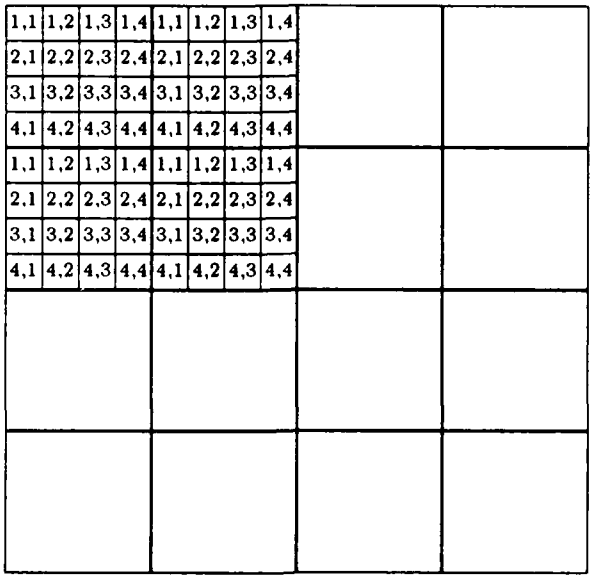


Figure 4: Interleaving of a linear system in a 4 x 4 processor grid.

For $k << N$ we can again approximate f as in section 3.1 to obtain the approximate total time

$$T_{P,R} \approx \frac{N^3}{3k}\omega + \frac{1}{2}N(N+2k)\tau + (N+k)\beta \quad \text{for} \quad k << N \tag{3.11}$$

Notice that the arithmetic time is the same as that of the broadcast algorithm. Incidentally, this means that when communication is very fast with respect to arithmetic, then the two algorithm will show the same performance. For the particular case where $k = N$, i.e., in the case of one row per processor, we have $f(N,k) = \frac{1}{2}(N-1)(N+2)$, and therefore an arithmetic efficiency of 2/3 is achieved just as for the broadcast algorithm.

An important difference between the above estimate and that of the broadcast algorithms is that the contribution of communication start-ups, i.e., the β term, is now linear in k and N while for the broadcast algorithm this contribution was of the form $O(\beta N Log_2 k)$. If β is large compared with the coefficients τ and ω, as is the case with Intel's iPSC for example, then this term may dominate the total time when $k = O(N)$. This is widely verified by our numerical experiments.

4. Grid algorithms

In this section we assume that the number of processors, denoted by k, is of the form $k = 2^n$, where n, the dimension of the cube, is even, and we define $\kappa \equiv \sqrt{k} = 2^{n/2}$. We consider the assignment of the linear system into the nodes of the hypercube that is the result of mapping the system into a $\sqrt{k}$ x $\sqrt{k}$ square grid of processors which is embedded into the hypercube. This is illustrated in Figure 4, where a label (i,j) in a block means that this block is mapped into the processor which in the row i and column j of the processor grid.

Similarly to row oriented algorithms we will consider broadcast and pipelined algorithms.

4.1. The broadcast grid algorithm

In the broadcast grid algorithm, we must now broadcast in two directions: we must send the elimination sub-rows to every processor in the same vertical line of processors of the grid, and the

sub-column of multipliers horizontally. Moreover, the multipliers must be computed before being moved. In other words the j-th step could be described simply as follows:

1. Broadcast the element a_{jj} vertically.
2. Compute the multipliers $a_{i,j}/a_{j,j}$ in the processors that hold the elements $a_{i,j}$.
3. Broadcast the sub-columns of multipliers horizontally and the elimination sub-rows vertically.
4. Proceed with the elimination process in each processor.

Remember that each column or row of processors in the embedded processor grid is a subcube of dimension $n/2$. Therefore step 1, which is a broadcast of a single element, takes the time $(Log_2\kappa)(\beta+\tau)$. Step 2 consists of at most $\lceil (N-j)/\kappa \rceil$ arithmetic operations and costs $\lceil (N-j)/\kappa \rceil \omega$.

Step 3 is a broadcast of at most $\lceil (N-j)/\kappa \rceil$ words in each direction. The two data transfers can be overlapped in the two horizontal and vertical subcubes of dimension $n/2$ and therefore the time required for the broadcast in step 3 is bounded from above by

$$2\left(\lceil (N-j)/\kappa \rceil \tau + \beta Log_2\kappa\right)$$

Note that as usual we take the largest time as given by the ceilings $\lceil . \rceil$ since these reflect the times taken by the processors that finish last.

Finally, step 4 consists of linear combinations of $\lceil (N-j)/\kappa \rceil$ rows of length $\lceil (N-j)/\kappa \rceil$ each and hence the time for this step is

$$\lceil (N-j)/\kappa \rceil^2 \omega.$$

Adding the above times and summing them over the steps $j = 1, \ldots, N-1$ we find the following the total time valid when N is divisible by κ

$$T_{B,G} = \frac{1}{3}N(q+1)(q+2)\omega + \frac{1}{2}N(q+1)\tau + \frac{3}{2}N\beta Log_2 k, \tag{4.1}$$

where here

$$q \equiv \frac{N}{\kappa}. \tag{4.2}$$

Observe that in the limiting case where $k = N^2$ the time to solve a linear system by this algorithm is dominated by the communication start-up time $\frac{3}{2}\beta N Log_2 N$. Therefore, this is not an effective method for this case since there are algorithms that realize Gaussian elimination in $O(N)$ time using $O(N^2)$ processors. The pipelined method described next is one such algorithm.

4.2. The pipelined grid algorithm

We now consider a pipelined Gaussian elimination algorithm for the grid mapping of Figure 4. It is assumed here that no pivoting is used. The generalization of the ring pipelined algorithm can be easily understood if, conceptually, we view each row of processors in the processor grid, as one processor which is part of a ring of κ processors. To implement the pipelined ring algorithm we observe that the multipliers must now be also moved from left to right in each row of processors. Thus we must pipeline the information in both horizontal and vertical directions. The implementation is a straightforward generalization of the pipelined ring algorithm. A processor typically receives a pivot row, then forwards it to the next processor down and a similar operation is done for the column of multipliers. As soon as both data have arrived the processor proceeds with the arithmetic. An important detail is that at the jth step, the multipliers are computed in the processors that contain the elements a_{ij}.

Looking at the first block-row of A, contained in processors $P_{1,*}$, the rightmost column of A, will receive the N/κ multipliers with a delay of

$$(\kappa - 1)(\frac{N}{\kappa}\tau + \beta) + \frac{N}{\kappa}\omega$$

i.e., the time to travel across $\kappa - 1$ processor and to form the multipliers in the leftmost processors. This assumes that each processor receives the column then sends it immediatly to its right neighbor before proceeding with the arithmetic.

Looking at the rightmost column of processors, we use again the fact that, as for the ring algorithm, the last row is critical, in the sense that it determines the total time of the algorithm. Every new step will consist of an arithmetic task whereby $\lceil (N-j)/\kappa \rceil$ sub-rows of size $\lceil (N-j)/\kappa \rceil$ each are combined with the pivot sub-row, and then sending the previously received pivot sub-row downward. Thus every new step j in the bottom rightmost proceesor will require the time

$$\lceil \frac{N-j}{\kappa} \rceil^2 \omega + \lceil \frac{N-j}{\kappa} \rceil \tau + \beta$$

Summing these times over the steps $j = 1, 2, \ldots N-1$ and adding the above starting delay we get the approximate time

$$T_{PG} = \frac{1}{6}N(q+1)(2q+1)\omega + \frac{1}{2}N(q+5)\tau + (N+2\kappa)\beta, \tag{4.3}$$

where again q is defined by (4.2). Observe that as noted earlier a time of $O(N)$ can be achieved.

5. Numerical experiments

As an experiment we tested the broadcast row algorithm and the pipelined ring algorithm on Intel's iPSC-d7, which is a hypercube of dimension seven. The linear system was taken so that its solution is known. More precisely we chose a system of size size $N = 127$, and the matrix

$$a_{i,j} = 1 \quad \text{for } j \neq i, \quad \text{and} \quad a_{i,i} = N + 1.$$

All components of the right hand side b are taken equal to $2N$, so that the exact solution is known to be $x = (1, 1, 1 \ldots 1)^T$.

We implemented a column oriented algorithm rather than a row oriented one. The linear system is therefore assigned by columns using interleaving, the right hand side being considered as the last column of the matrix. For the pipelined algorithm we used the Gray code to embed a ring $P_0, P_1, \ldots, P_{k-1}$. This was not done for the broadcast algorithm for which it is not needed.

We solved the linear system with the two methods varying the number of processors from $k = 2$ to $k = 128$ in each case. This can be easily done by varying the dimension of the iPSC. Thus the results shown are only for numbers of processors that are powers of 2. The comparison of the performances of two methods shown in Figure 5 confirms the prediction that pipelined methods are generally faster.

To check the accuracy of the models of performance analysis proposed in this paper and in other papers [9, 12, 18, 17], we compared the real times and the estimated times provided by our models. For the broadcast algorithm we used the non-optimal formula (2.4) instead the upper bounds (2.6), as it reflects the present state of the iPSC. The iPSC uses packets of length of 1KB: hence in our case the number of packets is always one. The communication start-up time was measured to be approximately 6.5 ms. The peak bandwidth of each channel is 10Mbits/sec which gives $\tau = 3.2\mu s$. However, various measures conducted elsewhere lead to the the actual measured

time $\tau \approx 8\mu s$, corresponding to less than half the optimal speed. As for the parameter ω, we used $\omega \approx 9.25 \times 10^{-5}$. This was obtained by using two processors and evaluating the number of operations performed as well as estimating the communication time which is then deducted. Thus this time for one pair of multiplication and one addition includes such times as memory fetches, loop up-dating and so on. The plot in Figure 6 shows the result for the broadcast algorithm while that of Figure 7 shows the result for the pipelined algorithm. The dashed lines in both plots are the estimates. Both are relatively accurate given the simplifying assumptions made to derive the models. Moreover, our experience is that in the pipelined algorithm the communication start-up varies substantially. For example, it seems that when a receive command is issued while a send command is still being executed, then β is increased, possibly doubled. In fact, as is shown in Figure 8, a much better agreement is obtained for $\beta = 10\ ms$, which roughly assumes that the above conflict occurs half of the time.

6. Discussion

In what follows we make a few concluding remarks and examine briefly some of the issues such as pivoting, that have not been addressed in this paper.

1. There are no particular reasons for using a broadcast algorithm. Plots of the theoretical times for various values of the parameters β, ω, τ, show the pipelined algorithm to be always either faster than, or very close to (for small k), that of the broadcast algorithm.
2. Similarly, grid algorithms seem to be generally superior to row or column oriented algorithms. Ring algorithms cannot be used for $k>N$ but this is only part of the story. Not only do pipelined grid algorithms have smaller communication times but also when $k = O(N)$ the arithemtic efficiency of the row algorithms deteriorates to 2/3. On the other hand since in this situation $\sqrt{k} << N$, an arithmetic efficiency of nearly one is achievable for the grid algorithms.
3. Partial pivoting is easy to implement for column or row mappings. For grid mappings, partial pivoting is no longer attractive and should be replaced by pairwise pivoting which is straightforward to implement [4].
4. An interesting conclusion is that the hypercube topology does not seem to do any better than the grid topology for Gaussian elimination. Hypercubes would be superior if partial or total pivoting were crucial for stability but the recent paper by D. Sorensen [22] seems to indicate that pairwise pivoting will always be good enough in practice.
5. Various methods for solving the resulting triangular system have been devised for various mappings, except grids in [9]. For reason of space we defer the study of these to another report. We should only point out that there are efficient ring algorithms for $k << N$ and grid algorithms for $\sqrt{k} << N$. However when $\sqrt{k} = O(N)$ then it may be a better idea to switch to the Gauss-Jordan method which avoids triangular systems solutions.

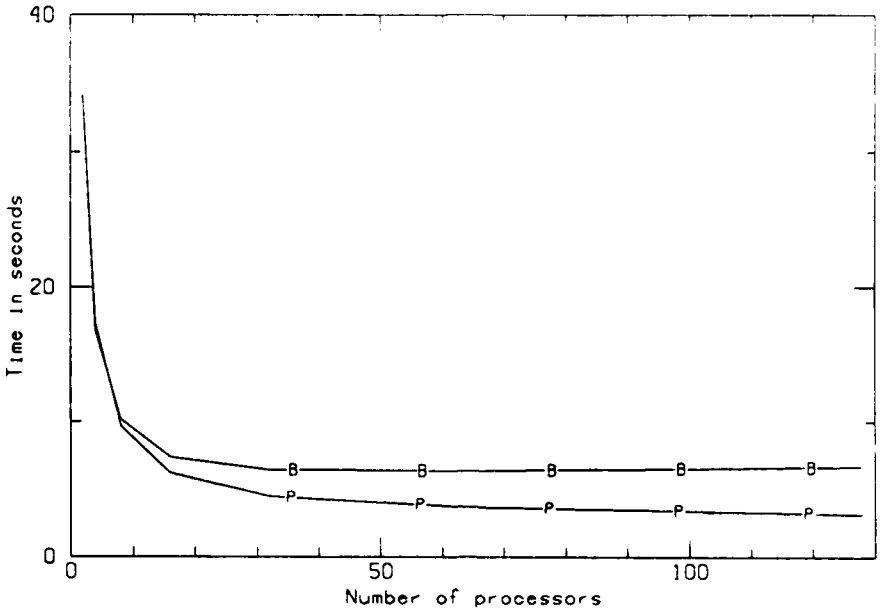


Figure 5: Comparison of the broadcast row algorithm (B) and the pipelined ring algorithm (P).

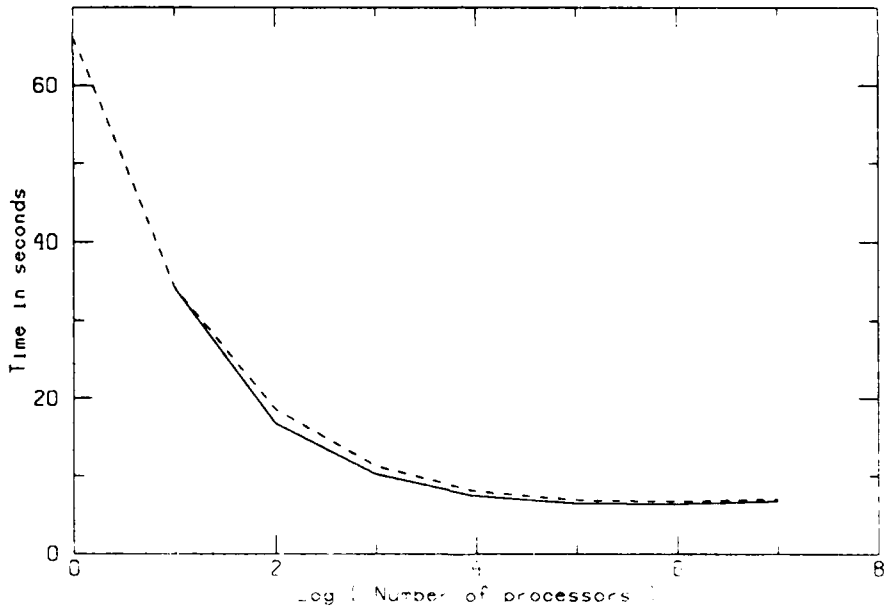


Figure 6: Comparison of the actual times (solid line) and estimated times (dashed line) for the broadcast row algorithm.

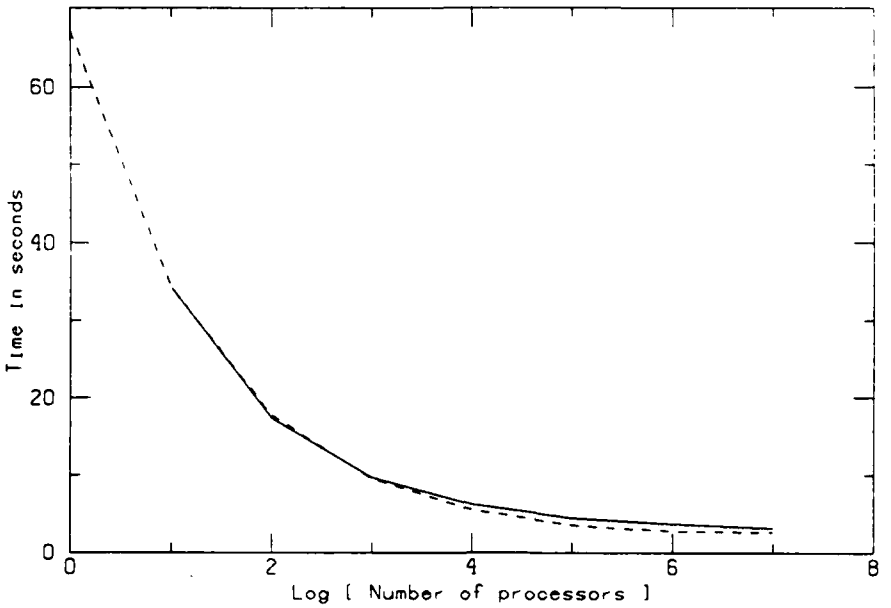


Figure 7: Comparison of the actual times (solid line) and estimated times (dashed line) for the pipelined ring algorithm.

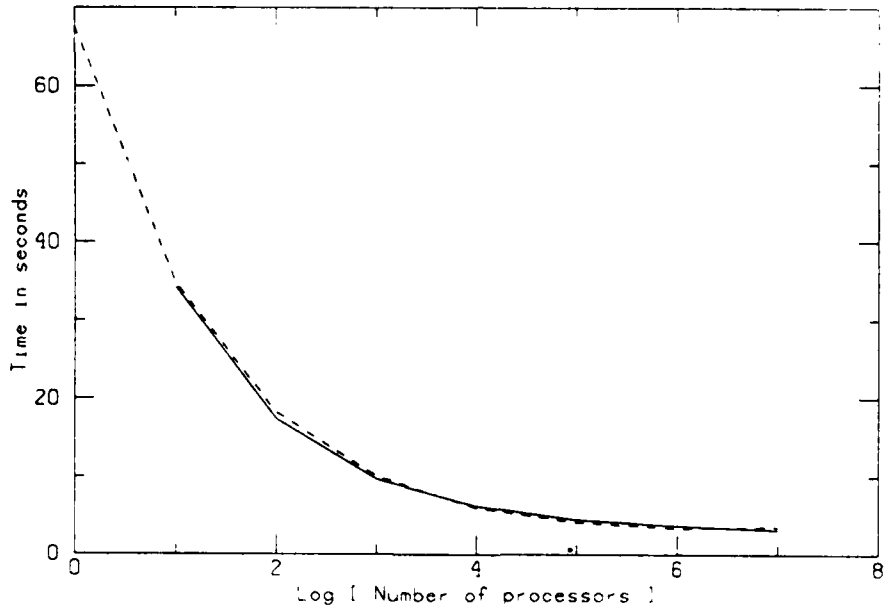


Figure 8: Comparison of the actual times (solid line) and estimated times (dashed line) for the pipelined ring algorithm, when $\beta = 10\ ms$.

References

[1] S.N. Bhatt, I.C.F. Ipsen, *How to imbed trees in hypercubes,* Research Report 443, Dept Computer Science, Yale University, 1985.

[2] L. N. Bhuyan, D.P. Agrawal, *Generalized Hypercube and Hyperbus structures for a computer network,* IEEE Trans. Comp., C-33 (1984), pp. 323–333.

[3] T.F. Chan and Y. Saad, *Multigrid Algorithms on the Hypercube Multiprocessor,* Research Report 368, Dept Computer Science, Yale University, 1985.

[4] J.J. Dongarra, A. Sameh and D.C. Sorensen, *Implementation of some concurrent algorithms for matrix factorization,* Technical Report ANL/MCS-TM-25, Argonne National Lab., 1984.

[5] G.A. Geist, *Efficient Parallel LU Factorization with pivoting on a hypercube multiprocessor,* Technical Report ORNL-6211, Oak Ridge National Lab, 1985.

[6] G.A. Geist, M.T. Heath, *Parallel Choleski Factorization on a Hypercube Multiprocessor,* Technical Report ORNL-6190, Oak Ridge National Lab, 1985.

[7] W.M. Gentleman, and H.T. Kung, On Stable Parallel Linear System Solvers, *Proc. SPIE (Real Time Signal Processing IV),* 1981, pp. 19–26.

[8] M.T. Heath, *Parallel Cholesky Factorisation in Message-Passing Multiprocessor Environments,* Technical Report ORNL-6150, Oak Ridge National Laboratory, 1985. Submitted to *Parallel Computing.*

[9] I.C.F. Ipsen, Y. Saad and M.H. Schultz, M.H., *Complexity of Dense Linear System Solution on a Multiprocessor Ring,* Lin. Alg. Appl., (1986).

[10] S.L. Johnsson, *Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures,* Technical Report YALEU/CSD/RR-361, Dept. of Computer Science, Yale University, September 1985.

[11] ————, *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures,* Technical Report YALEU/CSD/RR-367, Yale University, Dept. of Computer Science, February 1985.

[12] S.L. Johnsson, Y. Saad, and M.H. Schultz, *The Alternating Direction Algorithm on Multiprocessors,* Research Report 382, Dept Computer Science, Yale University, 1985.

[13] R.E. Lord, J.S. Kowalik and S.P. Kumar, *Solving Linear Algebra Equations on a MIMD computer,* Journal of the Association of the Computing Machinery, 34 (1983), pp. 103–117.

[14] C.A. Mead and L.A. Conway, *Introduction to VLSI Systems,* Addison Wesley, Mass.,USA, 1980.

[15] D.P. O'Leary and G.W. Stewart, *Data-Flow Algorithms for Parallel Matrix Computations,* Technical Report 1366, Dept Computer Science, University of Maryland, 1984.

[16] Y. Saad and M.H. Schultz, *Topological Properties of Hypercubes,* Research Report 389, Dept Computer Science, Yale University, 1985.

[17] ————, *Data Communication in Hypercubes,* Research Report 428, Dept Computer Science, Yale University, 1985.

[18] ————, *Direct Parallel Methods for Solving Banded Linear Systems,* Research Report 387, Dept Computer Science, Yale University, 1985.

[19] A.H. Sameh, Numerical Parallel Algorithms - A Survey, Lawrie, D. and Sameh, A.H. ed., *High Speed Computer and Algorithm Organization,* Academic Press, 1977, pp. 207–28.

[20] A.H. Sameh, D.J. Kuck, *On Stable Parallel Linear System Solvers,* JACM, 25 (1978), pp. 81–91.

[21] C.L. Seitz, *The Cosmic Cube*, CACM, 28 (1985), pp. 22–33.

[22] D.C. Sorensen, *Analysis of pairwise pivoting in Gaussian elimination*, Technical Report ANL/MCS-TM-26, Argonne National Lab., 1984.

END

DTIC

7-86